36

OBJECT-ORIENTED AND RELATIONAL DATABASES: A COMPARATIVE STUDY OF CONCEPTS AND APPLICATIONS

by

Sheila Azadeh Norcio

Major paper submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

APPROVED:

[illegible]
C.J. Egyhazy, Chairman

[illegible]
S. Antoy

[illegible]
E. Haddad

May, 1990

Blacksburg, Virginia

C.2

LD
5655
V851
1990
N672
c.2

OBJECT-ORIENTED AND RELATIONAL DATABASES:
A COMPARATIVE STUDY OF CONCEPTS AND APPLICATIONS

by

Sheila Azadeh Norcio

Committee Chairman: Csaba J. Egyhazy
Computer Science

(ABSTRACT)

The entity-relationship and extended-entity-relationship models are outlined. The concepts are used to introduce the object-oriented and relational models, operations, implementation, and application domains.

Extensions of relational databases are examined to uncover object-oriented techniques utilized. The object-oriented techniques include: addition of entity identifiers, addition of new types, expansion of the data manipulation and definition languages. Differences between object-oriented and relational databases are discussed. They include: schema design, encapsulation and instantiation, degree of use, addition of new types, integrity and security, version management, and distributed processing.

Computer-aided design (CAD), computer-aided software engineering (CASE), office information systems (OIS), and universities administrative systems (UAS) are selected for a comparative analysis of relational and object-oriented databases. A threshold model consisting of the following criteria is designed: schema evolution, encapsulation, ease of use and implementation, addition of new types, security and integrity, importance of versions, distribution, and use of object identifiers.

The relational and object-oriented databases used in the analysis are defined depicting their features as they relate to the criteria under analysis. The requirements for CAD, CASE, OIS, and UAS categories are defined as well.

Finally, the threshold model is used to determine the suitability of the two database models for each application category. It is then concluded that object-oriented databases fulfill the requirements of CAD, CASE, and OIS applications whereas the relational database fall unacceptably short. Object-oriented databases are judged marginally advantageous for the UAS application.

A description of the analysis and the conclusion is included.

TABLE OF CONTENTS

OBJECT-ORIENTED AND RELATIONAL DATABASES: A COMPARATIVE STUDY OF CONCEPTS AND APPLICATIONS

I. INTRODUCTION

While relational databases remain the front-runner in the category of commercial applications, their appropriateness to other areas of applications such as computer-aided design (CAD), office information systems (OIS), and computer-aided software engineering (CASE) has been contested. In the past few years, the object-oriented model has emerged as the likely candidate to fill the gap left by the relational model. While several types of object-oriented databases have been identified, a standard object-oriented model has not yet emerged. However, there are common components that form the basis for an object-oriented model. In this paper, we will look at what seems to constitute that common thread. [Carey 88]

Meanwhile, there has been tremendous activity to extend the relational model to address the new application categories. In general, the extensions aim at capturing more semantics in the database while maintaining the attractive features of the relational model such as its mathematically based structures.

This paper examines the ways in which the relational model is extended to capture more semantics by using object-oriented techniques. Further, the extended relational and object-oriented models is juxtaposed to determine whether critical differences exist that could affect the selection of one model over the other.

Lastly, the paper will examine four categories of applications: CAD, CASE, OIS, and UAS--representative of the typical commercial application-- to determine whether the object-oriented is more appropriate than the relational.

The remainder of the paper is organized in the following fashion: Section I is an introduction to the basic database concepts and the relational and object-oriented models; Section II is a survey of relational database extensions using object-oriented techniques. Section III consists of a comparison of the two models in meeting requirements of the application categories under study.

This section will present database concepts and terminology. First, the Entity-Relationship (ER) and Extended Entity-Relationship (EER) models are outlined. The ER and EER model are used as a platform to explain the relational and object-oriented models.

Basic concepts of typical relational and object-oriented databases will be defined. For each database, the model, typical operations, architecture, integrity, and application domain will be explained.

I.A. Database Concepts and Fundamentals

A database, regardless of the model it espouses must possess certain basic features which will characterize it as a database.

- A database management system (DBMS) has a non-trivial model and language. That is, the DBMS understands some structure on the data (the data model) it contains, and provides a language for manipulating that structured data.
- A DBMS can represent relationships between entities, the relationships can be named, and the language can query those relationships.
- A DBMS provides a persistent and stable store. By persistent we mean that data are accessible past the end of the process that creates them. By stable, we mean that data have some resiliency in the face of process failure, system failure, and media failure. This resiliency is due to a recovery mechanism, which writes information about changes to the database to secondary storage, and uses this information to make corrections to the data after a failure.

- A DBMS permits simultaneous use of the database by multiple users, although not necessarily of the same data items at once.
- The address space of a DBMS is not constrained by limitations in the physical processor. Thus, the size of a database should not be limited by the amount of main memory, or by the address range of virtual memory.
- A DBMS can help to ensure the correctness and consistency of the data it contains, by enforcing integrity constraints, which are statements that must always be true for data items in the database. [Zdonik 90]

Two of the models used in the database field are the ER and EER models. The ER, EER, relational, and object-oriented models each have produced different types of databases. However, each of the more recent models have borrowed concepts and techniques from the predecessors. In this section, the ER and EER models are explained to lay the foundation for the definition of relational and object-oriented databases.

I.A.1. The Entity-Relationship Model

The entity-relationship (ER) model has been most successful as a tool for communication between the designer and the end-user during the requirements analysis and conceptual design phases because of its ease of understanding and its convenience in representation. [Teorey 86]

In defining the ER model, initially, Chen proposed three classes of objects: entities, attributes, and relationships. Entity sets were the principal distinguishable objects about which information was to be collected and usually denoted a person, place, thing, or event of informational interest. Attributes were used to detail the entities by giving them descriptive properties such as name, color and weight. Finally, relationships represented real-world associations among one or more entities. [Teorey 86]

The relational database design approach uses both the ER model and the relational model in successive stages. It benefits from the simplicity and ease of use of the ER model and the structure (and associated formalism) of the relational model. [Teorey 86]

There are two types of attributes: identifiers and descriptors. The former is used to uniquely distinguish among the occurrences of an entity, whereas the latter is used to describe an entity occurrence. Entities can be distinguished by the "strength" of their identifying attributes. Strong entities have internal identifiers that uniquely determine the existence of entity occurrences. Weak entities derive their existence from the identifying attributes of one or more "parent" entities.

Relationships have semantic meaning, which is indicated by connectivity between entity occurrences (one to one, one to may, and to may), and the participation in this connectivity by the member entities may be either optional or mandatory.

I.A.2. The Extended Entity-Relationship Model

The EER model introduces category abstraction into the ER model. The introduction results in two additional types of objects: subset hierarchies and generalization hierarchies. The subset hierarchy specifies possibly overlapping subsets, while the generalization hierarchy specifiers strictly non-overlapping subsets. [Teorey 86]

A generalization hierarchy occurs when an entity (which we call the generic entity) is partitioned by different values of a common attribute. The generalization object is called an "IS-A" exclusive hierarchy. The introduction of the subset and generalization hierarchies allows for the capture of more real-world meaning in the database. A subset hierarchy is the case in which every occurrence of the generic entity may also be an occurrence of other entities that are potentially overlapping subsets. These concepts have later been used by the object-oriented databases in the development of the model. [Teorey 86]

The generalization hierarchy implies that the subsets are a full partition, such that the subsets are disjoint and their combination makes up the full set. The subset hierarchy implies that the subsets are potentially overlapping. [Teorey 86]

I.A.3. Schemas and Instances (Evolution and Extensibility)

It is important to recognize that a database is a dynamic, evolutionary body of information. The current contents of a database are termed an instance of a database, or equivalently, a snapshot of the database is its instance at that moment.

The schema consists of the database entities and the relationships among them. As we will see later one major difference between the relational and object-oriented model is the ease with which the model accommodates dynamic changes to the scheme of the database by the user. The evolutionary quality of a database is termed extensibility.

There is a great difference in the way schemas and changes to them are managed by relational and object-oriented databases. As we will see in section I.B. object-oriented databases allow for dynamic changes of the schema and manage versions of entities as they change. Relational databases are on the other hand incapable of such evolution and change control.

I.A.4. Constraints and Semantics

Disallowed entities or relationships may be excluded by defining restrictions called constraints on the entity set. Constraints are expected to be true for any structure within the database schema. [Tsichritzis 82]

Semantic data modeling aims at capturing the meaning of data while still preserving data independence. Semantic is captured with the goal of handling database operations in a more intelligent manner. The semantic of data may be represented with the generalization, aggregation, and association relationships. [Codd 79]

While both object-oriented and relational databases handle constraints on entities and capture meaning of data, they differ greatly in the extent of which they accomplish those goals. We will see in section I.B how successful each database is in maintaining the integrity of the database and the modeling of the entities being represented.

I.B. Relational Databases

Relational databases are based on the relational model proposed by Codd in 1970 which primarily addressed the issue of data independence. Relational database management systems

(RDBMS) have successfully been commercially used for some years.

I.B.1. The Relational Model (Representation)

The relational model was introduced to correct the problems of data independence and data inconsistency present in the databases of the day. Those databases lacking in the properties targeted by Codd were primarily based on the network or hierarchical models. [Codd 70]

Codd proposed that the conceptual view of data should remain unchanged when the internal representation of data is modified. He also proposed the relational model as a basis for treating the issues of redundancy and consistency of relations. The following is a description of the relational model as it has been implemented in RDBMSs.

The mathematical concept underlying the relational model is the set-theoretic relation, which is a subset of the Cartesian product of a list of domains. A domain is simply a set of values. The Cartesian product of domains D1,D2,...,Dk, written D1xD2x...xDk, is the set of all k-tuples (v1,v2,...,vk) such that v1 is in D1, v2 is in D2, and so on [Ullman 82].

A relation is any subset of the Cartesian product of one or more domains. The members of a relation are called tuples. It helps to view a relation as a table, where each row is a tuple and each column corresponds to one component. The ordering of rows is irrelevant and all rows are distinct. The columns are often given names called attributes which are the names of the corresponding domains. The ordering of columns is significant corresponding to the order of the domains D1,D2,...,Dk. However, Codd proposed that users deal not with relations which are domain-ordered, but with relationships which are their uniquely identifiable domain-unordered counterparts, at least within any given relation. [Codd 70] [Ullman 82]

The set of attribute names for a relation is called the relation schema. The collection of relation schemas used to represent information is called a database schema, and the current values of the corresponding relations is called the (relational) database. [Ullman 82]

An attribute (or combination of attributes) in a relation scheme has values that can uniquely identify each component of that relation scheme. Such an attribute is called a primary key. The primary key corresponds to the descriptor identifier of the ER model. It is important to note that the primary key in the relational database is value-based, which, as we will see later, could restrict the freedom of the user in changing its value.

One of Codd's primary objectives in the proposal of the relational model was the elimination of the need for user knowledge of the machine representation of data. The relational model provides maximal independence between programs and machine representation and organization of data. This feature of the relational model was its major distinguishing factor from other data models of the day and it has been an inherent part of its implementation. [Codd 70]

I.B.2. Operations

While the insertion, deletion, and modification of tuples are necessary operations on the database; it is however the query operations that represent the challenging aspects of a data manipulation language.

Query languages for the relational model are based on two broad classes:

1. Algebraic languages
2. Predicate calculus languages

Given the mathematical foundation of the relational model, operations on the database are easily formulated and optimized. Since relations in the relational model are sets, set operators such as union, intersection, and set difference are applicable to those relations. Other useful operations are the intersection, quotient, join, projection, selection, cartesian product, and the normal join. [Ullman 82] [Codd 70]

I.B.3. Architecture

The physical database design, a nontrivial task for large and complex databases, requires a good understanding of how the database will be used and the frequency of different operations. Also of fundamental importance to relational database is the management of primary keys. [Date 86]

Relational DBMS's use different file organizations. Among the used implementations are the heap file organization, hashed, and index-sequential schemes. Naturally, the goal of the physical design is to speedup data access, retrieval, and modification.

I.B.4. Integrity

The integrity of databases are maintained by providing the DBMS with a mechanism for expressing constraints on relations such that errors may be detected and prevented. DBMSs should be able to enforce two essentially different kinds of constraints. One set of restrictions on relations depend on the semantics of domain elements. These restrictions depend on understanding what components of tuples mean. The second set depend only on the equality or inequality of values. Additionally, as it is impossible for databases to be devoid of redundancies, it is critical that consistency be maintained between different copies of the data.

I.B.5. Applications

The relational data model was introduced at a time when business systems such as employee information systems and financial systems were the typical database applications. Since the introduction of object-oriented models other database categories have been targeted which were not satisfactorily addressed by the relational model and its predecessors. In section III, we will discuss the suitability of the relational model versus that of the object-oriented model as they treat each different category of applications.

I.C. Object-Oriented Databases

The fields of programming languages, artificial intelligence, and software engineering have contributed towards the development of the object-oriented data model. The first appearance of the notion of an object as a programming construct was in Simula, a language for programming computer simulations. The object-oriented model like object-oriented programming languages prescribes to the idea of encapsulation and reusability. Similar to relational databases, object-oriented databases possess the basic requirements of a DBMS described in section I.A. However, the foundations of the object-oriented are entirely different.

I.C.1. The Object-Oriented Model and Operations

An object-oriented model is based on the encapsulation of objects. An object is an abstract machine that defines a protocol through which users of the object may interact . The protocol of an object is typically defined by a visible interface consisting of a set of messages (i.e. operations) with typed signatures. The objects realization (i.e. its data structures and the implementation of its operations) are hidden from the user. The object interprets the message and carries out the operation if parameters are of the correct types. [Nierstrasz 89] [Zdonik 90]

A message is implemented by the object method. A message name can be overloaded. In object-oriented systems, operations may be polymorphic, that is they work on a variety of object types.

The same operation maintains its behavior transparently for different argument types. Class inheritance is closely related to polymorphism. The same operations that apply to instances of a parent class also apply to its subclasses. Polymorphism enhances reusability of code.

Every object is an instance of some class. A class is therefore a template for its instances. Classes are typically arranged in a directed graph, with the edges connecting superclasses to their subclasses. In object-oriented databases users are provided with facilities to define their own classes and instantiating their own objects. An object class specifies a set of visible operations, a set of hidden methods that implement operations. [Nierstrasz 89]

An important feature of the hierarchy of classes is the implementation of inheritance rules. Inheritance varies depending on what is inherited and when and how inheritance takes place. Class inheritance is often represented as the fundamental feature that distinguishes object-oriented models from other models.

Object-oriented models provide for database schema evolution or extensibility through mechanisms for creating new types. Although the idea of creating new types is not new to the database field, the view that a type is really an abstract data type that encapsulates its implementation is rather original.

Object-oriented data models are also characterized by the ability to make references through an object identity. The identity of the object should remain invariant across all possible modifications of the object's value. The relational database model is however value-based. Since entities are identified by their keys, the value of the key may not be changed. On the other hand, in the object-oriented model the identity which can never be changed is not part of the object's realization.

Similar to the aggregation concept in the EER model, another important concept in an object-oriented database is the idea of collections --IS-PART-OF-- which serve as a way to aggregate related objects. Like the EER model, the object-oriented model allows for sharing of objects. This parallels the subset hierarchy of the EER model. Query operations are performed over collection of objects.

As new types are introduced into the database, new algebra for queries are added. Unlike relational databases in which query languages remain static and can consequently allow for optimization, object-oriented databases do not lend themselves as easily to optimization. However, the dynamic evolution of the database schema is what makes extensibility a possibility. We will discuss the advantages and disadvantages of object-oriented versus relational databases as they pertain to query languages and their efficiency in section II.

In object-oriented databases a relationship consists of a correspondence between objects. Relationships or properties may be objects themselves and as such they will be subject to a protocol which will describe their behavior.

I.C.2. Architecture

Object-Oriented databases are typically implemented with an interface to an object manager which is in charge of handling the activities of objects as they relate to disk storage and retrieval, and buffering of data. The degree of semantics built into the storage manager is an important architectural design issue.

The intelligence built into the object manager can have significant efficiency implications. For example, the object manager can improve speed by clustering collections of objects in the same physical area.

I.C.3. Integrity

Integrity constraints are predicates on the state of the database that the database management system is responsible for enforcing. Unlike most other DBMSs which typically support schema-level constraints, object-oriented databases have the capabilities to allow for greater granularity with integrity constraints definition at instance-level or for that matter at a superclass level. [Zdonik 90]

I.C.4. Applications

CAD, CASE, OIS, and scientific applications are among databases that object-oriented databases support typically not addressed by other DBMSs. However, object-oriented databases can also address the application areas previously handled by other DBMSs. We will attempt to discover the ease and efficiency with which object-oriented databases can manage different application areas when compared to relational databases. However, it is important to note that while relational DBMSs have matured and are in commercial use, object-oriented databases are still in developmental stage and are just beginning to reach the mass market. As such, our discussion is comprised of comparisons of theories, techniques, and concepts and does not represent the similarities or differences of marketed products.

II. RELATIONAL DATABASE DESIGN USING THE OBJECT-ORIENTED APPROACH: TECHNIQUES AND ISSUES

When introduced in 1970, the relational data model primarily addressed a single category of applications: the typical business systems. Those applications included among others UASs. However, other application areas such as computer-aided design (CAD), computer-aided software engineering (CASE), and office information systems (OIS) which have more complex data requirements have since taken center stage in the database field. While the addition of more semantics to the database is motivation enough for the extension of the relational database, it is, however, to address the needs of these new categories of database applications that numerous paths in meeting those demands have been pursued. The primary goal of initiated efforts consist of allowing for more flexibility and control on the part of the user in order to represent the application domain more accurately.

One approach attempted in expanding the application domain of the database systems has been to extend the current relational data models. The introduction of new scalar types, structured types, and complex data types and the capability to define adequate data manipulation for the new types have been the area of focus.

Meanwhile, the object-oriented model has been introduced as an alternate approach to meeting the more rigorous data requirements. Object-oriented databases have brought the application programs and the database systems closer. In doing so, object-oriented databases have successfully addressed the more complex data requirements. This has been possible in part by mechanisms allowing for the creation of new data types using the basic data types embedded in the primary database system. Along with the creation of new data types, object-oriented databases allow for the definition of new operations on those types.

In this section, we will discuss the techniques used in extending the relational data model to accomplish the task of expanding its application domain. The methods used will be evaluated to determine the extent of their use of object-oriented techniques in accomplishing that goal. On the other hand, we will analyze areas in which the two models differ, disclosing issues which could be a determining factors in the choice of one database over the other.

II.A. Techniques

The main limitation of the relational model is its semantic scantiness, that often prevents relational schemas from modeling completely and expressively the natural relationships and mutual constraints between entities. Since Codd introduced the relational model many extensions have been proposed with the intent to enlarge the application domain addressed by relational databases. Originally, the relational model focused on typical business applications (e.g. inventory or personnel systems). However, the focus has shifted to include more complicated categories such as CAD applications. In extending the relational model, a wide variety of approaches are used which also include object-oriented concepts. In this section, a survey of the most prevalent extensions is introduced while comparing the degree of adherence to object-oriented concepts. [Zaniolo 90] [Codd 70]

Codd asserts that a meaning oriented data model stored in a computer should enable it to respond to queries and other transactions in a more intelligent manner. Such a model could also be a more effective mediator between the multiple external views employed by application programs and end users on the one hand and the multiple internally stored representations on the other. Proponents of RDBMSs seeing the need for more functionality (i.e. as provided by object-oriented models) have attempted to extend the relational databases while remaining as close to the original relational model as possible. [Codd 79]

Stonebraker asserts that a complete extended type system should allow [Stonebraker 86a]:

1. The definition of user-defined data types
2. The definition of new operators for these data types
3. The implementation of new operators for these data types
4. Optimized query processing for commands containing new data types and operators.

In general, extended relational systems support higher-level modeling features, such as are found in semantics data models, plus some capabilities found in object-oriented databases (OODBs): adding new scalar types, set- or array-valued attributes and DML stored in the database. [Zdonik 90]

In our survey, the various extension techniques will be evaluated for the ease in which the database is expanded without disrupting ongoing work. However, our analysis of extensions to the relational model is examined limited to object-oriented techniques. [Codd 70]

II.A.1. What parts of the database is being extended?

The primary extension to relational databases has been the addition of entity identifiers or surrogates. This is an important deviation from the original value-based attribute identifiers.

Further, new types have been introduced into the database. These new types build upon the primitive types of the system and can vary in complexity from scalar types to complex types. Along with the introduction of the new types, operators have been introduced to handle them.

II.A.1.a. Addition of entity identifiers

Almost universally all new models extending the relational database model introduce the idea of an entity or object identifier, otherwise called surrogate. Surrogates are system assigned and never change. Their value cannot be changed by users. Each entity or object is assigned a unique ID and all references are made via the ID. Object ID's are completely independent of changes in data value and physical location. [Codd 79] [Blaha 88]

Codd asserts that the need for unique and permanent identifiers for database entities such as employees, suppliers, etc., is clear. User-defined and user-controlled primary keys in the relational model which were originally intended for this purpose pose difficulties. There are three problems in employing user-controlled keys as permanent identifiers for entities [Codd 79]

1. The values of user-controlled keys are determined by users and must therefore be subject to change by them.
2. Two relations may have user-controlled keys defined on distinct domains and yet the entities denoted are the same.
3. It may be necessary to carry information about an entity either before it has been assigned a user-controlled key value or after it has ceased to have one.

Since surrogates uniquely identify entities or objects, primary keys are no longer necessary for entity identification. Surrogates undertake the role of the primary keys. Subsequently, the value-based keys, traditionally known as the primary keys become another set of descriptive attributes. Thus, the user will no longer have to invent arbitrary keys for purposes of identification. Additionally, the primary key for relationship tables would be one or more IDs from participating objects.

There are however extended relational DBMSs that introduce more semantics into the model without the addition of object identifiers. The POSTGRES data model introduces new abstract data types such as data of type procedure while still maintaining the value-based primary keys of the relational DBMS as the identifying component of the tuple. [Rowe 90]

II.A.1.b. Addition of new types (Abstract data types)

The most important extension mechanism employed is the addition of new types to the primitive system types. The relational model has been extended with abstract data types including user-defined operators and procedures, relation attributes of type procedure, and attribute and procedure inheritance. These mechanisms can be used to simulate a wide variety of semantic and object-oriented modeling constructs including aggregation and generalization, complex objects with shared subobjects, and attributes that reference tuples in other relations. [Rowe 90]

II.A.1.b.i. Scalar types

In relational databases the domain of attributes form a small set basic types. Extended and extensible relational databases allow for definition of new types based on types already existing in the DBMS. In general, the introduction of new types into the database facilitate a more accurate representation of semantics inherent in the application model. [Schwarz 86]

The capability to add new data types has been one of the primary areas of extension of relational databases. This capability is launched with the addition of user-defined scalar types (i.e. atomic).

In the original relational model attributes can only be atomic (e.g. integer, floating point, or boolean). However, extended relational DBMSs allow the user to extend the system by adding new atomic types using an ADT definition facility. [Rowe 90]

Although user-defined scalar types are represented by the base types, different operators, implemented by extensions to the primary database system, need to be defined for them. Scalar types facilitate more thorough semantic checking. [Schwarz 86]

II.A.1.b.ii. Structured types

This concept generalizes to hierarchically-structured types. As in the case of scalar types, the definition of a new structured type would include new operators for the type that implemented. Arrays are another construct for structuring data that are not part of relational databases. However, they are widely used in general purpose programming languages. In order to eliminate this aspect of the impedance mismatch between the database and the programming language, such structured types as arrays are also added to the collection of types supported by the extended relational DBMS. [Schwarz 86]

Among the built-in data types are unbounded varying length arrays of fixed types with an arbitrary number of dimensions. Variable length arrays are provided for applications that need to store large homogeneous sequences of data. The availability of set-valued attributes also adds to the conciseness and expressivity of the relational database. [Rowe 90]

II.A.1.b.iii. Complex objects (aggregation, generalization, and inheritance)

In order to address the more complex application domain, relational databases have added complex objects to their repertoire. Complex objects are defined as data structures that are composed of an arbitrary collection of records from one or more relations. Relational databases have been extended by allowing relations with attributes such that their domain is a set of relations. A relation inherit attributes from its parent(s). Relations may inherit from more than one parent. [Schwarz 86]

Complex objects are generally represented through generalization and aggregation relationships. Mechanisms to define this type of relationships are provided in the DDL and DML. Generalization allows one to refine the structure of entities and add detail as needed. One can choose the proper level of abstraction for each context. The resulting design is robust and extensible. Smith and Smith define generalization as an abstraction in which a set of similar objects is regarded as a generic object. There are two aspects to this notion: instantiation and subtype. Both are forms of specialization, and their inverses are forms of generalization. The extensional counterpart of instantiation is set membership, while that of subtype is set inclusion. Associated with a generalization hierarchy is the property inheritance rule. [Blaha 88] [Codd 79]

A solution to allow for complex objects is to allow records to have fields that themselves are relations. The primary database system must be able to retrieve these embedded relations and must provide a mechanism for scanning or querying them. [Schwarz 86]

Each object class maps directly to one table. All object fields become attributes of tables. Complex objects are represented through the generalization, aggregation, and association relationships. A generalization or is-a relationship partitions a class into mutually exclusive subclasses. The same object is being represented at each level of the generalization. Note that OMT supports multiple inheritance. Aggregation is an assembly-component or a-part-of relationship. Aggregation may be multilevel and recursive. For example, a data structure may recursively refer to itself. [Blaha 88]

A generalization relationship has one superclass table and multiple subclass tables. The connection between the two tables is made via the common object ID. Aggregation relationships are represented depending on the multiplicity of the relationships. Aggregation relationships are broken down into existence-dependent and free-standing aggregations which would determine the representations. Associations are mapped to distinct tables.

Attribute types can be of type procedure. The extended relational DBMSs query languages allow for values of type procedure to be stored in an attribute. The value of an attribute of type procedure is a relation. The value may include tuples from different relations. [Rowe 90]

II.A.1.c. Query language (additional operators)

Zero or more operators can be implemented for new The precedence level which is required when several user defined operators are present and precedence must be established among them. The operators are then compiled to be used. [Stonebraker 86a]

By introducing new types such as set-valued attributes, the extended relational DBMSs provide accordingly for a number of new operators such as the following set operators [Zaniolo 90]

= (set) equals

!=	(set) does not equal
>	properly contains
>=	contains
<	is properly contained in
<=	is contained in

II.A.1.d. Data access method

If database are to be used to store the new kind of data manipulated by new applications, they must also support the specialized access methods needed to retrieve this data with the performance such applications require. [Schwarz 86]

A DBMS should provide a wide variety of access methods, and it should be easy to add new ones which will efficiently support user-defined data types. In general an access method is simply a collection of procedure calls that retrieve and update records. An access method implementor utilizing interfaces to the log manager, the concurrency control manager, and the buffer manager. [Stonebraker 86]

Access methods fall into three general classes. Access methods that are the tightly bound to the primary system. Both the structure that implements the access method and the information stored in that data structure are supplied by the primary database system. Another access method uses one of the data structures provided by the primary database system to map keys to record-location information, but the data stored in the structure is either derived or interpreted in a nonstandard way. Still another access method is one for which the data structure used to store access information is not managed by the primary database system. The data stored in the structure may also be derived or interpreted in a nonstandard ways. Exogenous access methods are useful for applications that can only access data efficiently using specialized search structures.

Current relational database systems use indexes for fast access to records in a single relation. An extension could be used to manage indexes that could improve the performance of multi-relation queries.

II.A.1.e. Addition of functions

The addition of functions is based on the object-oriented methods which brings the programming language and the database closer together. It provides for operations other than those produced for by the DML on the entities.

In extended relational DBMSs, user-defined procedures may be written in a conventional programming language and can be used to move a computation from a front-end application process to the back-end DBMS process. [Rowe 90]

II.A.2. Where are the extensions incorporated?

As new types have been introduced into the relational databases, accordingly the data manipulation and data definition languages have been extended to manage new types.

II.A.3.b. DDL

Once the capability for definition of new types have been added to the relational data models, so have mechanisms been provided in the data definition language.

The DDL provides the facilities for the definition of new data types, the operators applicable to them, and the new access methods.

The DDL provides support for the new notions of entities with surrogates, aggregation, generalization, and set-valued attributes. Thus allowing users to define new types as needed.

II.A.2.b. DML

With the addition of user-defined operators and access methods, extended relational DBMSs have altered their primitive DML to support the new extensions. The DML supports access methods for the newly defined types, such as new indexing mechanisms. It should also be able to implement the new query operators defined by the DDL.

II.B. Issues

Extensions to relational databases have expanded their application domain. There are, however, pronounced differences from object-oriented databases. In this section, we will focus on those diversities and analyze how they could be used in addressing application requirements.

II.B.1. Schema design

With traditional DBMSs the design phase results in the translation of the model into the database schema which is specified in the DBMS using the DDL. After the construction of the schema and after it has been fine-tuned in the design process, the schema is compiled into machine usable form. As users create applications and store data for the applications, the data must conform to templates specified in the schema. The schema is static in the sense that modifications often require a system shutdown and/or reorganization of the database, which can be very expensive for a large database. This large overhead effectively prevents the use of dynamic or interactive schema in traditional DBMSs. For many new database applications such as CAD, CASE, and OIS there is need to consider interactive systems with schema. [Diedrich 89]

Existing conventional database systems allow only a few types of schema changes. This is because the applications they support (conventional record-oriented business applications) do not require more than a few types of a schema changes, and also the data models they support are not as rich as object-oriented models. [Kim 89a]

While extended relational databases may allow for addition of new types, they are however static once defined. The schema is compiled and cannot be changed dynamically.

II.B.2. Encapsulation and instantiation

Encapsulation represents a software engineering methodology that makes a sharp distinction between the specification of a module and the code and data structures that are used to implement this specification. Encapsulation is a powerful system-structuring technique in which a system is made up of a collection of modules, each accessible through a well-defined interface. The abstract data-type approach defines the interface by a set of strongly typed operation (also called message) signatures. It also requires that each type define a representation (an instance of some existing data type) that is allocated for each of its instances. This representation is used to store the state of the object. Only the methods implementing operations for the objects are allowed to access the representation, thereby making it possible to change the representation without disturbing the rest of the system. [Zdonik 90]

In the extended relational data models, new types may be constructed, but the underlying representation of the type remains visible. While encapsulation is an inherent part of object-oriented databases, it is absent in relational DBMSs.

Instantiation is perhaps the most basic object-oriented reusability mechanism. Every programming language provides some built-in data types (like integers) that can be instantiated as needed. Objects may be either statically or dynamically instantiated. [Diedrich 89]

Limited instantiation capabilities have been added to extended relational DBMSs with the introduction of generalization, specialization, and inheritance concepts.

II.B.3. Ease of use and implementation

In general, relational databases allow for limited dynamic modifications, such as addition of attributes, but are predominantly static in incorporating new mechanisms. In addition, newly defined operators have to be compiled to be recognized by the data manipulation language. Likewise, data access methods have to be compiled to be recognized by the DML.

Object-oriented databases allow for the run-time definitions of types and methods. Thus, system shutdown is not a necessary condition of schema evolution. This allows the user more freedom to manipulate the database and the adaptability of the database to the application domain. This is possible because types, methods, etc. are also objects contained within the database.

II.B.4. Manipulation of complex objects/new types

Many applications require the ability to define and manipulate a set of objects as a single logical entity for purposes of semantic integrity, and efficient storage and retrieval. Collections of objects augment the semantic integrity of an object-oriented data model through the notion of dependent objects. A dependent object is one whose existence depends on the existence of another object, and is owned by exactly one object (i.e. IS-PART-OF). The definition of a collection of objects as a composite object also offers an opportunity to improve the performance of a database system. A composite object may be used as a unit for clustering related objects in the database. This is because, when an application accesses the root object, it is often likely to access all (or most) dependent objects as well. Thus, it is advantageous to store all constituents of a composite object as close to one another as possible on secondary storage. Further, in a multiuser environment, a composite object may be used as the unit of locking to reduce the system overhead associated with concurrency control; that is a composite object may be locked as a unit, rather than requiring a lock for each component of a composite object. [Kim 87]

A problem in the relational model is that a complex object typically must be decomposed into tuples over several relations, and there is no data item that represents the object as a whole. Hence, some approaches concentrate on treating the complex object as a single unit for purposes of query, copying, locking or physical placement. Other projects have focused on the ability of complex objects to share subparts. In most relational database systems, there is no sharing of records between aggregate structures, or a record can participate in a fixed number of aggregates. The introduction of surrogates is a mechanism for support of shared subparts. [Zdonik 90]

While extended relational DBMSs have added new types which include complex objects, they still have to superimpose the tuple structure on them introducing efficiency problems.

II.B.5. Integrity and security (classes and type checking)

Security refers to the protection of data against unauthorized disclosure, alteration, or destruction; integrity refers to the accuracy or validity of data [Date 86]. In other words:

- security involves ensuring that users are allowed to do the things they are trying to do.
- integrity involves ensuring that the things they are trying to do are correct.

Various kinds of failures (program, processor, media) and violations (consistency, access, typing) can compromise the integrity and security of a database. A database system must be able to cope with failure by restoring the database to a consistent state, and should prevent violations from occurr- ing. By program failure we mean that an application program may fail to complete due to a run-time error. Processor failures, consist of the processor storage management functio- ning improperly. The database must remain intact in the face of the mentioned failures. By media failure we mean disk flaws such as bad sectors which may cause committed data to be lost. Protecting against the first two kind of failure requires that copies of objects on disk be kept consistent and updated carefully, while the third type of failure requires replica- tion of objects on disk. [Bretl 89]

A DBMS can help to ensure correctness and consistency of the data it contains by enforcing integrity constraints, which are statements that must always be true for data items in the database. Some common kinds of constraints are domains for fields in records, which specify the range of legal values for a field; keys, which say that certain data values serve to identify uniquely items in a collection; and referential integrity constraints, which assert that a reference in one data item indeed leads to another data item. [Zdonik 90]

Almost all constraints that current DBMSs, including relational DBMSs, support are schema-level constraints. They are defined relative to entities in the schema, and apply to all instances of a type or to all elements of a collection. In relational databases typechecking is performed when writing the transaction to the database. However, in object-oriented databases typechecking is performed dynamically. In relational databases referential integrity which is directly related to functional dependencies are extremely difficult to maintain. This is due to the syntactic nature of the model. However, object-oriented databases which capture more semantic do consequently manage functional dependencies more naturally. [Zdonik 90]

Versioning also introduces another dimension of integrity control into the database. Propagation control is an important aspect of version management which includes control of the consistency of data over the timing of the version update and the objects that are updated.

Transactions in business data processing are the unit of atomicity, recovery, integrity, and visibility by other users.

The trend in transaction support for CAD, CASE, and OIS is to provide these capabilities at different levels of granularity. For design applications in particular, it is useful to have constraints that apply to a single object. [Zdonik 90]

II.B.6. Versioning

There is a general consensus that version control is one of the most important functions in various data-intensive application domains, such as CAD systems and OIS dealing with compound documents. Users in such environments often need to generate and manipulate with multiple versions of an object, before selecting one that satisfies their requirements.

A version can be created from a relation or a snapshot. Updates to a version do not modify the underlying relation and updates to the underlying relation will be visible through the version unless the value has been modified in the version. A merge command is provided that will merge the changes made in a version back into underlying relation.

A selection of a set of mutually consistent versions is normally referred to as configuration management. In a CASE environment, a configuration is one version of each of the modules in a system such that set of versions produces a consistent system. The task of selecting this set of versions is referred to as configuration management. [Zdonik 90]

The notion of automatic propagation of values is related to the problem of version and configuration management. This information propagation or triggering is called percolation. When an object is updated or deleted, or a new version of the object is created, some or all of the objects that have referenced it may become invalid, and thus need to be notified of the change. [Zdonik 90]

Version management is of critical importance to versionable databases. These databases introduce the idea of transient and working (or generic) versions to control update propagation. They also present rules to manage the stability of the database. These rules are based on invariants that have to hold true before and after the update is applied to the database.

For the most part relational databases do not provide versioning which is a significant requirement of CAD and OIS. Versions are a way to record the history of an object. A given version can have several successors as well as several predecessors. The latter case is used for merging competing versions into a single agreed-on combined form. Few extended relational DBMSs allow users to save and query historical data and versions. By default, data in a relation is never deleted or updated. Conventional retrievals always access the current tuples in the relation. [Zdonik 90] [Stonebraker 86b]

II.B.7. Distributed databases

Modern database systems are frequently distributed, meaning that the data is kept at multiple sites. A common reason for installing a distributed system in the first place is simply that there may not exist any single machine with adequate capacity for the application. The various sites are connected into a communication network. A user (end-user or application programmer) at any site can access data stored at any site. In many ways of a distributed system represents a partnership among a set of independent but cooperating centralized system, rather than as some kind of monolithic and indivisible object. It is normal for the links to be relatively low speed relative to the speed at which a file can be read off of a disk. The consequence of this assumption about communication is that the transfer of data between computers becomes a bottleneck, and most of the issues unique to distributed systems concern ways of dealing with this bottleneck. [Date 86] [Ullman 82]

- special optimization techniques
- concurrency control problems

One opportunity for better performance in an object-oriented framework derives from the fact that programs (i.e., methods) are objects. As such, they can be moved around in the distributed database just like any other object. In performing a computation or processing a query, the system has the choice of moving the data to the programs, or of moving the programs to the data. [Zdonik 90]

To simplify programming and to preserve data independence, a major objective of distributed systems is to provide what is usually called location transparency - meaning that users should not need to know at which site any given piece of data is stored, but should be able to access the entire database as if it were stored at their own local site: it simplifies the logic of application programs, and it allows data to be moved from one site to another. [Date 86] [Zdonik 90]

A second objective in distributed databases is to support data fragmentation. A system supports data fragmentation if a logical object can be divided up into pieces (fragments) for physical storage. In relational databases a fragment could be any arbitrary subrelation that is derivable from the original relation by means of restriction and projection operations. In object-oriented databases, the methods may be separated from the objects instance variables. A system that supports data fragmentation should also support fragmentation transparency - i.e. users should be able to behave in all cases as if the relation were not fragmented at all (data independence again). Relations are easy to fragment, ant the fragments are easy to recombine. [Date 86]

III. APPLICATION SUITABILITY: RELATIONAL VERSUS OBJECT-ORIENTED

In this section we will look at four different categories of applications and their database needs. Computer-aided design (CAD), computer-aided software engineering (CASE), and office information systems (OIS) have of late been the focus of database developers. Each category has its own particular set of requirements. To fulfill the database gap created by these applications new database models have been introduced that aim at meeting those requirements. This section will compare the suitability of relational or object-oriented in meeting the needs of CAD, CASE, OIS, and UAS. This comparison is aimed at exhibiting the applicability of the database models in meeting the new application categories as well as the typical commercial applications.

We will define the criteria based on which the appropriateness of object-oriented databases versus relational databases will be judged as they apply to each application category. The criteria included in the threshold model will be described in section III.A. They are chosen based on the differences between the two models.

Section III.B will put forth the relational and object-oriented databases that will be used to evaluate the applicability of each model to the application categories. Neither the relational nor the object-oriented databases outlined represent a particular commercial database but rather an abstract representation of concepts common to most databases implementing each model.

Section III.C will enumerate the requirements of each application category. The requirements are stated without regard for any particular database model. In fact, it may be possible that those requirements will not be satisfied by either of the two models under consideration.

Finally, in section III.D, the paper addresses the suitability of the models to each of the application categories. For each application, the relative suitability of the relational and object-oriented models will be examined.

III.A. The Model

In order to determine the suitability of relational versus object-oriented databases for each of the application categories under study, a threshold model on which the analysis is based was created. The model addresses criteria that may or may not be a feature of either database.

The model consists of eight criteria where the relational and object-oriented databases differ. In some instances, the difference is due partly to marginal diversities in the feature. In other instances, the disparity is more extreme such as the absence of the feature in one or both of the two databases. When a feature is absent in the model, such as encapsulation in relational database, it is not automatically a disadvantage for that database. It is, rather, compared against the requirements of the application category. If the application requirements demand the feature, then it is considered a disadvantage for the applicability of the database.

Figure I is the format for the representation of the suitability of the relational and object-oriented databases as ranked based on each criteria. The "application category" in the table will in each case be replaced by the specific application category such as CAD, CASE, OIS, or UAS. The title for each row represents the criteria used to evaluate the applicability of the database model. The data for each model is recorded under their appropriately titled columns of the table.

The values will range from 0 to 10 expressing not suitable at all to perfectly satisfactory. An example of a zero ranking would be the case where an application requiring version management cannot obtain the feature with the proposed database model (i.e. the feature is absent in the model). A value of 10 will represent the case where the application has a strong need for a feature and the database model completely satisfies that requirement. The ranking reflects the importance of the feature to the application category and the ability of the database to satisfy the needs of that

Application Category	Relational	Object-Oriented
Schema Evolution		
Encapsulation		
Ease of Use		
New Types		
Security/Integrity		
Versions		
Distribution		
Object Identifiers		
Total		

Figure I - The Treshold Model

application category. However, if a feature is not needed by an application category it will be marked not applicable.

III.A.1. Schema evolution

Traditionally databases make a very strong distinction between instances and schemas. Instances are in the database, whereas schema information is stored in the data dictionary. Many database systems have two different languages to deal with instances and schemas. The Data Manipulation Language (DML) deals with operations on instances. The Data Definition Language (DDL) deals with operations, mainly creation, on schemas. Traditional databases allow very little flexibility for evolution of their classes. Schema evolution is very restricted. Relational systems are better than other system in that they sometimes permit adding attributes. However, dropping attributes or moving them to other relations is seldom permitted (see Section II.B.1). [Tsichritzis 88]

Object-oriented systems, however, can manipulate instances and classes (schemas). Classes are in fact genuine objects. Existing commercial database systems do not provide such facilities. They do provide, however, extensive facilities for class definitions in the database dictionary. It is conceivable that these facilities can be made available, and integrated as database operations. However, in doing so database systems will lose some of the simple user interfaces.

The great advantage of relational systems is based on the relative few, very basic and very clear operations. [Tsichritzis 88]

The relationships supported between database classes, whether they be relations or record types, etc., are quite restricted. They may be statically defined between classes, as in entity-relationship schemas. Relational systems, on the other hand, allow many relationships, but they are completely syntactic, based on contents and operations like joins. [Tsichritzis 88]

This criteria will be used to evaluate the application category requirements for schema evolution and the degree of suitability of the database model. The schema evolution criteria will be used to determine schema extensibibity required by the application and how well each of the two database models accommodate that need.

III.A.2. Is encapsulation a necessary condition?

We will consider the encapsulation criteria from two different perspectives. First, the ability of the database to modify its instance variables and methods without affecting related objects. Second, the success of the database in reusing code and inheritance control.

Encapsulation is a technique for minimizing interdependencies among separately-written modules by defining strict external interfaces (see Section II.B.2). A module is encapsulated if clients are restricted by the definition of the DBMS to access the object only via its defined external interface. To maximize the advantage of encapsulation, one should minimize the exposure of implementation details in external interfaces. A DBMS supports encapsulation to the degree that it allows minimal external interfaces to be defined and enforced. One characteristic of an object-oriented DBMS is whether it permits a designer to define a class such that its instance variables can be renamed without affecting clients. [Snyder 90]

This issue raises the fundamental question of the purpose of inheritance. One can view inheritance as a private decision of the designer to "reuse" code because it is useful to do so; it should be possible to easily change such a decision. alternatively, one can view inheritance as making a public declaration that objects of the child class obey the semantics of the parent class, so that the child class is merely specializing or refining the parent class. [Snyder 90]

In relational databases entities are accessed by reading instance variables (i.e. primary key), defying the encapsulation rule. Most DBMSs, however, promote inheritance as a technique for specialization and do not permit a class to exclude an inherited operation from its own external interface. [Snyder 90]

Since this criteria may only be present in object-oriented databases it could possibly eliminate the applicability of the relational databases entirely. However, if the application category has no need for this database feature, it will then be of no significance to the evaluation of the two databases (i.e. N/A). The encapsulation criteria will also focus on how well methods or functions are embedded within the database.

III.A.3. Ease of use and implementation

Database systems traditionally provide very few generalized types (i.e., record types, relations, etc.). As a result they can provide a small number of very general operations for queries and updates on the database objects. The operations are the same regardless of the semantics of the objects involved. Queries and updates on employees, cars, accounts etc. utilize the same operations. Thus, the operations are easy to learn. [Tsichritzis 88]

Object-oriented systems require that all objects provide their own set of operations, with some sharing through object classes and inheritance mechanisms. In addition, the methods can be logically complex. Most of the work in object-oriented databases deals with extending databases operations to accommodate particular object types. The extensions take two forms. First, complex objects can be defined, thus dealing with structural complexity within objects. Second, operations specific to object classes can be defined. Multiple inheritance can be used to define new classes that share operations and attributes with existing classes.

Object-oriented systems provide facilities to manipulate instances and classes. Classes are in fact genuine objects. Relational DBMSs do not provide such facilities. They do provide, however, extensive facilities for schema definitions in the database dictionary. However, the great advantage of relational systems is based on the relative few, very basic and very clear operations. [Tsichritzis 88]

This criteria will be determined by two variables:

1. ease of learning
2. ease of extensibility

The ease of learning will be measured based on the simplicity of the query language. The extensibility feature, in turn, stresses ease of implementation. This feature is important in evaluating how flexible the DDL is in dynamically mapping the logical schema of the database to the database schema.

III.A.4. Is there a need for new types?

This criteria will determine whether the primitive data types of the system are sufficient for the application category under scrutiny. It will determine how well new types may be added to the database (see section II.B.4). The new types typically include scalar types, structured types, and complex types (aggregation and specialization/generalization). In particular, complex objects are of special interest.

Many applications require the ability to define and manipulate a set of objects as a single logical entity. We define a composite object as an object with a hierarchy of classes to which the objects belong as a composite object hierarchy. A composite object hierarchy captures the IS-PART--OF relationship between a parent class and its component classes, whereas a class hierarchy represents the IS-A relationship between a superclass and its subclasses. [Banerjee 87]

III.A.5. Are security and integrity issues?

Due to the encapsulation of the methods within the object, typechecking for object-oriented databases is no longer limited to static typechecking or only at write time. The timing of the integrity checks also becomes a factor.

Since our database models both have solid integrity and security controls, this criteria will evaluate the granularity at which they are applied. The significance of the granularity is studied as it applies to the application categories.

We will also examine the temporal aspects of integrity mechanisms. By this, we mean the time at which integrity constraints are applied to the entities.

III.A.6. Importance of versions

Versions are variations of the same object that are related to the history of their derivation. Most application systems in the CAD and OIS domains require version control. A common requirement of these applications is the desire to preserve alternative states for a particular entity. Users in such environments often need to generate and experiment with multiple versions of an object before selecting one that satisfies their requirements. [Banerjee 87] [Fishman 89]

Many things have properties that vary with time, or with respect to other parameters. Does it make sense to version such things as employees, departments, products, projects, schedules, inventories, documents, etc.? [Kent 89]

With this criteria, we will evaluate the need of each application for versions. Further, the manner in which updates to versions are managed (propagation control) will be examined.

How is update managed? Is it necessary to pre-declare an object type versioned? Versioning is generally associated with a change in the state of an object. But what constitutes the "state" of an object? [Kent 89]

What are the application areas that require versioning support? Electronic design, mechanical design, software engineering, document management, and what else? [Kent 89]

III.A.7. Degree and/or importance of distribution

Traditional databases deal with distribution, if at all, by hiding it. A distributed database is a logically integrated, physically distributed database. The network is not visible, and we seldom have a notion of context, either as a geographic location, i.e., a workstation, or as a logical context (see Section II.B.7). [Tsichritzis 88]

Object-oriented systems need a strong definition of context. First, we believe that objects should be aware of where they are. Physical location in the network may affect their behavior. Second, objects, or collection of objects, may encapsulate beliefs, and we therefore need a context to define a boundary. Third, objects' behavior may be affected by their context. Sometimes they should even directly inherit methods from their context. A simple example is a text object that inherits formatting characteristics of globally coordinated object managers where objects are managed by local object managers but in a completely integrated and transparent manner. [Tsichritzis 88]

We are therefore faced with an interesting dilemma. On one hand distributed databases strive to provide a uniform globally integrated database. On the other hand object-oriented systems seem to require a strong notion of context. To what extent the two can co-exist depends a lot on how objects are mapped into databases. [Tsichritzis 88]

This criteria will determine whether an application category needs a distributed system and as such how well it can be satisfied by either the relational or object-oriented databases.

III.A.8. Use of object of identifiers

When we model real-world objects with some particular purpose in mind, however, we only include some subset of that object's description in the model. This subset may not be complete enough to capture the object's uniqueness. In some cases uniqueness is external (e.g., an object is unique if it has some local attribute values and belongs to a different set, or is related to a different object). If the concept of identity is built into a DBMS, then an object's uniqueness is modeled even though its description is not unique. [Khoshafian 90]

Database systems utilize object identifiers internally for implementation purposes. These identifiers are to be visible and available for manipulation by the user in older database systems. In the relational model, and in some relational systems tuples do not have a visible identifier. They are identified by their contents, via primary or secondary keys. [Tsichritzis 88]

There are a host of powerful data modeling concepts which have been introduced in database models. One of these concepts is the need to model arbitrarily complex and dynamic objects with versions. A more specific need in this representation is the ability to distinguish objects from one another regardless of their content, location or addressability, and to be able to share objects. Object identity enables us to realize this goal. [Khoshafian 90]

There are several problems with identifier keys which are due to the fact that the concepts of data value and identity are mixed (see Section II.A.1.a). The solution calls for built-in support for identity in the language which is independent of its external descriptive data, so that the system can provide a strong notion of identity in the representation. [Khoshafian 90]

Several researchers have argued for a temporal data model. The reason is that most real-world organizations deal with histories of objects, but they have little support from existing systems to help them in modeling and retrieving historical data. Strong support of identity in the temporal dimension is even more important for temporal data models, because a single retrieval may involve multiple historical versions of a single object. Such support requires the database system to provide a continuous and consistent notion of identity throughout the life of each object, independently of any descriptive data or structure which is user modifiable. This identity is the common thread that ties together these historical versions of an object. [Khoshafian 90]

To deal with this, database languages usually provide the capability to map the user's conceptual schema onto an internal schema, which describes the way that data is actually stored. An internal schema may have multiple copies of the conceptual schema and may further partition the attributes of an object of the conceptual schema. Some way of relating these multiple copies and attribute partitions to the same conceptual object is needed. The object's identity provides a convenient way of doing this.

The most powerful technique for supporting identity is through surrogates. Surrogates are system-generated, globally unique identifiers, completely independent of any physical location. Surrogates provide full location independence. [Khoshafian 90]

This criteria will be used to determine how static the identifiers of objects are and how important it is to allow the freedom to change primary key values. This criteria will examine the need for the reintroduction of identifiers into databases. It will also examine the effect of object identifiers on the realization of complex objects.

III.B. The Databases

The relational and object-oriented databases depicted in this section represent an abstraction of common features present in the corresponding commercial or prototypical databases currently in use. We assume that both databases have the basic requirements of a database as enumerated in section I.A.

The databases are further outlined to express their differences as used in evaluating the degree of suitability to each application category.

III.B.1 The relational database

We will use Ullman's definition of the relational model as the basis for the analysis. The relational database is a distributed multiuser system which is based on the relational model as defined by Ullman. Thus, it is based on the premise of entities represented as tuples. It provides for the typical relational operations such as joins, projections, etc. (see section I). It possesses a DDL which is based on the system defined types and does not allow for type extensions.

Operations are constrained to the system defined types as well. [Ullman 82]

(1) Schema Evolution: dynamic schema evolution is limited to addition of new attributes.

(2) Encapsulation: is not available.

(3) Ease of use and implementation: the set-theoretic basis of the DML renders the relation database easy to use. In addition, the model has the capability of being conceptually represented in table form thus making it easy to comprehend.

(4) New types: The relational database is limited to its primitive atomic types (integer, floating point, fixed-length strings, etc.).

(5) Integrity: is maintained when writing to the database. Integrity constraints are applied at relation schema level.

(6) Versioning: not available.

(7) Distribution: is supported by allowing for multiple sites and fragmentation.

(8) Object Identifiers: in relational databases, entity identifiers are the value-based primary keys. Thus, unique identifiers or surrogates are not utilized.

III.B.2. The object-oriented database

The object-oriented database represents a collection of features common in the currently operational or proposed object-oriented databases. The features under scrutiny in the threshold model are further explained.

(1) Schema Evolution: the object-oriented database allows for changes to its hierarchy of classes as well as to the classes themselves. It also accommodates active data such that an object can trigger an action upon being accessed. Changes to the class definitions include adding and deleting instance variables and methods. Changes to the class hierarchy include creation and deletion of a class, and alteration of the IS-A relationship between classes. [Kim 87]

Changes to the class hierarchy can be broadly categorized as (1) changes to the contents of a class, (2) changes to an hierarchy, and (3) changes to a class. [Banerjee 87]

1. Changes to the contents of a class
    - 1.1 changes to an instance variable
    - 1.2 changes to a method
2. Changes to an hierarchy

2.1 make a class S a superclass of a class C

2.2 remove a class S from the superclass list of a class C

2.3 change the order of superclasses of a class C

3. Changes to a class

3.1 add a new class

3.2 drop an existing class

3.3 change the name of a class

(2) Encapsulation: is a necessary requirement for object-oriented models (see section I).

(3) Ease of Use and Implementation: the object-oriented database is conceptually more complex than the relational model. The result of this complexity is that object-oriented databases are harder to use and learn than relational databases. However, the schema of the object-oriented database is more flexible and therefore lends itself more easily to the implementation of complex data objects.

(4) New Types: can be implemented by building on the concept of a metatype. The object-oriented database is therefore highly flexible in providing for new types.

(5) Integrity and Security: are implemented at the object level.

(6) Versioning: the object-oriented database provides for accessing versions of an object's state, and for assembling configurations of consistent versions of objects. [Zdonik 90].

(7) Distribution: the object-oriented database allows for distribution and fragmentation.

(8) Object Identifiers: are an inherent part of the object-oriented database. In object-oriented systems object identifiers are very important for two reasons. First, identifiers provide a permanent handle for objects that may evolve or move, in much the same way that file names hide the fact that a file's contents and physical location may change. Second, if an object's contents are properly encapsulated, they cannot be expected to provide a means for identification. [Tsichritzis 88]

III.C. Definition of the Application Categories

In this section, the requirements of each of the four application categories under study will be enumerated. These requirements will later be used in section III.D to evaluate the appropriateness of the relational and object-oriented databases.

III.C.1. CAD

CAD applications are unlike typical commercial environments in some important respects, necessitating a distinct approach to their modelling [Bapa Rao 86]:

1. Designs are developed by evolution; details are added incrementally to the design. Versions, or evolutionary alternatives of the design, must be maintained.
2. Designs are implemented by re-using other, previously defined, designs, leading to a component hierarchy.
3. A design often represents an integration of multiple views (or representations), each denoting some aspect of that design (e.g., the logical and physical views of a VLSI chip). Uniform descriptions of component structures and views must be maintained along with multiple versions of those components and views ("configuration management").

(4) CAD applications, such as VLSI circuit design, typically requires a group of designers to complete a complex design by closely interacting among themselves and dynamically sharing design data. This necessitates a distributed database environment. [Korth 90]

(5) CAD applications require the capability, to define, store, and retrieve as a single unit a collection of related objects known as a composite object. A composite object explicitly captures and enforces the IS-PART-OF integrity constraint between child and parent pairs of objects in a hierarchical collection of objects. Further, it can be used as a unit of storage and retrieval to enhance the performance of a database. [Kim 87]

III.C.2. CASE

A software environment is an application which requires the management of highly interconnected data. Modern environments attempt to provide a facility for managing the design, construction, testing, use, and eventual reuse of software. One of the most important requirements of a software environment is providing a central store for managing the myriad of objects that makeup a software project and keeping those objects up-to-date in the face of the many changes made over the lifetime of a project. [Hudson 89]

(1) As modern software environments will most likely be used in distributed workstation applications, this facility is viewed as crucial. It will be necessary to allow different users at different machines to

configure their own environments privately and share information. [Hudson 89]

(2) Software environments include complex data types such as programs, requirement specifications, milestone reports, configurations, documentation, and many other. These types are often defined in terms of each other, and need to be broken down into categories. For example a configuration is made up of a number of instances of the type program; source and object modules might be viewed as subtypes of type program. The idea is to provide a database of tool that would serve as the central repository of an environment, and to allow the sorts of derived information needed in an environment to be specified with as little additional code as possible. [Hudson 87]

(3) The ability to extend the type structure is necessary to allow users of the software environment to dynamically add new tools such as debuggers and compilers. [Hudson 87]

(4) An efficient rollback and recovery mechanism, which provides the framework for the recall of versions. It is of particular importance that versions not be represented as largely redundant objects, as objects in a software environment are likely to be quite large. [Hudson 87]

(5) Software environments typically deal with highly structured and interrelated objects. A primary example of this is of course computer programs, but software environments may also wish to deal with objects involving the management and control of an overall software development project. The sorts of object generally included in descriptions of existing and proposed environments include: software components and software dependencies, versions, documentation, requirements, milestone reports, test data, verification results, bug reports, etc. Note that "software components" which are themselves highly structured and interrelated entities as only one element of this list. Because of the complexity of the interrelationships defined in this model, its is essential that the consistency of the database is maintained automatically. [Hudson 87]

(6) Software environments, unlike most applications, deal with entities which change dramatically over time. The ability to retrieve and manipulate multiple versions of programming entities can be crucial to the programming process. In addition, we need the ability to manipulate versions and version streams as objects in themselves in order to support configuration management tools within the system. [Hudson 87]

(7) As described, software environments typically deal with data that is interrelated in such a way that changing one piece of data can have effects on many other data items. The data type "milestone" within an environment typically models the scheduled and expected completion times of a software component. One milestone may depend on another, and changing the expected completion date for one milestone may have effects that ripple throughout the expected completion dates for other milestones in the system. changing a milestone is an instance of a simple modification which affects the consistency of the database. If the expected completion date of a milestone is changed without also updating all the milestones that directly or indirectly depend on it, the database will be inconsistent an incorrect. [Hudson 87]

III.C.3. OIS

Office applications have characteristics that distinguish them from conventional data processing applications. Some of these characteristics have to do with the character of the applications themselves and some have to do with the character of the data on which these applications operate. [Zdonik 84]

(1) An office database should be a "total" information resource in that it should be capable of storing data of many arbitrary types. Users of such a system should be able to store conveniently their documents and graphics objects in the same logical storage space as their more traditional record-oriented data. [Zdonik 84]

(2) All of the many information types that will be stored in the database of an office workstation must be usable together effectively. It must be possible to be able to create connections dynamically between them as well a to treat them differently in different contexts. Many of the objects that occur in an office environment are built out of other objects. This kind of structure can be seen in reports that contain chapters and the chapters that further contain paragraphs. [Zdonik 84]

(3) Objects frequently need to reflect new understandings or conditions. Not only does the content of these objects change, but often the basic structure will change as well. Consider the many revisions that a typical final report will undergo.

(4) The objects in an office are often used in many different ways. The semantics of an object can depend on how it is being used. A chapter in a final report might be printed differently than the

same chapter in a working paper. Incremental development of new object types never stops. An office environment constantly produces exceptions to the previous way of expressing things. New definitions of what constitutes a report are always being discovered.

(5) OIS applications require an environment facilitating electronic mail. Electronic message passing takes place in a distributed environment.

(6) Calendars and schedules represent another aspect of OIS applications. Events in a calendar may trigger reactions in other entities in the office.

Office activities have been described as largely event-driven and semistructured, with a high level of parallelism requiring synchronization and coordination. [Weiser 89]

III.C.4. UAS

The UAS is a representative of the typical commercial business system. This is the type of application that has successfully been addressed by relational databases. However, we are interested in discovering whether an object-oriented database might better satisfy the requirements of such a system.

(1) The UAS will be operating from different sites, however, the information must be available across campus.

(2) The data descriptions of the UAS can be expressed as a (relatively) small number of object types, whose structures are (relatively) slow to evolve.

(3) The system needs to represent people with different roles (i.e. employee, student-employee, professor, etc.).

III.D. Database Suitability

This section will evaluate the suitability of relational and object-oriented databases as defined in section III.B for each of the application categories outlined in section III.C.

Each criteria in the model is ranked for both databases. The ranking does not represent absolute empirical data, it rather illustrates the disparity or similarity in satisfying the application requirements.

III.D.1. CAD

Figure II represents the collection of the evaluation of each criteria for CAD applications. Each ranking is further explained as follows.

(1) Schema Evolution: CAD designs are dynamic by nature (see requirement one in section III.C.1). The relational database schema evolution, limited to tuple addition, is essentially unsatisfactory. On the other hand, the object-oriented database can directly address this requirement with its dynamic schema evolution.

(2) Encapsulation: As stated in section III.B.1, encapsulation is missing from the relational database. However, it is a basic component in object-oriented databases (see section III.B.2). CAD applications have a strong need for this feature for satisfying the reusability notion in their environment.

(3) Ease of Use and Implementation: In CAD applications, the capability to reflect the design environment is crucial. The ranking reflects the fact that while the relational database has an easier user interface, it is, however, the capability to represent the application (ease of extensibility) that takes precedence.

(4) New Types: As stated in requirement five in section III.C.1, there is a need to create and manipulate complex objects. This feature is absent in the relational database as we have defined.

CAD	Relational	Object-Oriented
Schema Evolution	0	10
Encapsulation	0	10
Ease of Use	1	9
New Types	0	10
Security/Integrity	2	8
Versions	0	10
Distribution	10	10
Object Identifiers	0	10
Total	13	77

Figure II - The Treshold Model for CAD Applications

(5) Security/Integrity: The unit of design in CAD applications being an object, integrity and security constraints need to be applied at the object level.

(6) Versions: Requirement three in section III.C.1 defines the need for CAD versions. The relational database does not have the capability, while the object-oriented database is well suited to this need.

(7) Distribution: is relatively well-handled by both databases.

(8) Object Identifiers: The requirement for representing multiple view of the same entity (see requirement three) and the need for versions (requirement one) necessitate the availability of object identifiers present only in the object-oriented database.

III.D.2. CASE

Figure III represents the ranking of the criteria for CASE applications. The following further explains the disparities and/or similarities between the two databases.

(1) Schema Evolution: Similar to CAD, CASE applications require the capability to evolve due to their inherent design nature (see requirement three).

(2) Encapsulation: Reusability is one of the most important needs of CASE applications (see requirement two).

(3) Ease of Use and Implementation: In CASE environments, this criteria dwells in the capability to manage change (i.e. configuration management) and the myriad of components in a software. The object-oriented database addresses this need with its schema evolution and new types. The ease of use is measured by evaluating the ease with which CASE components can be managed.

(4) New Types: Entities in CASE applications may be as complex as a debugger (see requirement three in section III.C.2). The relational database is incapable of addressing this need.

(5) Security/Integrity: By allowing for versions, the object-oriented database may address the rollback and recovery mechanism stated in requirement four. In addition, the capability to provide for derived data and triggers, the object-oriented database can address the need to maintain the consistency described in requirement five.

CASE	Relational	Object-Oriented
Schema Evolution	0	10
Encapsulation	0	10
Ease of Use	0	10
New Types	0	10
Security/Integrity	0	10
Versions	0	10
Distribution	10	10
Object Identifiers	0	10
Total	10	80

Figure III - The Treshold Model for CASE Applications

(6) Versions: Requirement six undoubtedly describes the need of CASE applications for version management.

(7) Distribution: As described in requirement one, software applications will most likely be used in a distributed environment.

(8) Object Identifiers: The highly complex, variable, and interrelated data and the need for versions dictates the use of object identifiers.

III.D.3. OIS

Figure IV represents the ranking of the threshold model criteria for OIS applications. The following further explains the ranking of the two databases.

(1) Schema Evolution: Requirement three in section III.D.3 outlines the need for schema evolution for OIS applications.

(2) Encapsulation: Reusability is an important factor in OIS applications (see requirement four in section III.C.3).

(3) Ease of Use and Implementation: Similar to CAD and CASE applications, OIS applications require a great

OIS	Relational	Object-Oriented
Schema Evolution	0	10
Encapsulation	0	10
Ease of Use	2	10
New Types	0	10
Security/Integrity	2	7
Versions	0	10
Distribution	10	10
Object Identifiers	0	10
Total	14	77

Figure IV - The Treshold Model for OIS Applications

deal of flexibility. Thus, ease of extensibility is the important factor.

(4) New Types: Based on requirement two, OIS applications require to represent complex objects (i.e. objects defined in terms of other objects). The object-oriented database satisfies this requirement.

(5) Security/Integrity: Again, in OIS applications integrity needs to be maintained at the object level (e.g. a document).

(6) Versions: OIS applications have a strong need for version management (e.g. revisions to a document).

(7) Distribution: The electronic mail described in requirement five in section III.C.3 points to the need for distribution support.

(8) Object Identifiers: Again, similar to CAD and CASE applications, OIS applications require object identifiers.

III.D.4. UAS

Figure V represents the ranking of the model criteria for the UAS. The following outlines the applicability of the two databases.

(1) Schema Evolution: The UAS is rather static. The schema is stable and well established.

Universities	Relational	Object-Oriented
Schema Evolution	N/A	N/A
Encapsulation	N/A	N/A
Ease of Use	9	1
New Types	0	5
Security/Integrity	7	7
Versions	N/A	N/A
Distribution	10	10
Object Identifiers	0	10
Total	26	33

Figure V - The Treshold Model for the Universities System

(2) Encapsulation: N/A

(3) Ease of Use and Implementation: Extensibility is not a critical factor in the UAS. Since, the relational database is easier to use, we show a higher ranking in this criteria.

(4) New Types: Requirement three in section III.C.4 speaks to the notion of specialization/ generalization which can be represented by the object-oriented database.

(5) Security/Integrity: This requirement can be served equally well by both databases.

(6) Versions: The UAS does not have a distinct need for versions.

(7) Distribution: Both databases address equally well the distribution requirement explained in section III.C.4.

(8) Object Identifiers: Object identifiers are needed to maintain uniqueness despite changes to attributes representing the entities (i.e. change of department name).

IV. CONCLUSION

In general, we can assert that relational databases fall dramatically short for the CAD, CASE, and OIS applications with the more demanding requirements. However, in the UAS application category where relational databases have been used with great success, there may also be room for improvement by using object-oriented databases. However, the improvements acquired are marginal and may not justify the move if other criteria such as performance and/or cost are also considered.

The need for object identifiers is universal across applications. It is important that the identity of entities remain intact despite changes in attributes. The evolution of relational databases has led to the inclusion of object identifiers or surrogates in the DBMS, hence, correcting this shortcoming.

Entity versions are explicitly needed by CAD, CASE, and OIS applications. However, typical commercial applications such as the UAS could also benefit from the inclusion of a version management feature in the database. For example, a course taught in different semesters, could be represented as different versions of the same entity.

V. SUMMARY

Relational databases are still widely used in commercial applications. Extensions to those databases, such as addition of surrogates (entity identifiers), have corrected apparent shortcomings. At this time, it is unlikely that object-oriented databases will replace those well established databases. Rather, object-oriented concepts have been added to relational databases including aggregation and generalization mechanisms.

On the other hand, object-oriented databases have penetrated application areas, such as CAD, CASE, and OIS where relational databases have not been able to accommodate. While object-oriented databases are still in their infancy, they are rapidly maturing and reaching the mass market. Object-oriented databases can also meet requirements of the typical commercial applications. However, since they do not add a noticeable amount of functionality over relational databases (due to the more modest application requirements) they do not demonstrate a more advantageous front.

In the end, at this time, the two databases seem to serve different application categories. However, it is not clear, whether in the future the separating line will remain as clear.

VI. VITA

January 3, 1954 - Born - New Orleans, Louisiana

ACADEMIC

1976 - B.A., National University of Iran, Teheran, Iran

1980 - A.A., Northern Virginia Community College, Alexandria, Virginia

1982 - CPA, Virginia

PROFESSIONAL EXPERIENCE

1980-1982 - Accountant, Washington Gas Light Company, Washington, D.C.

1982-1985 - Internal Auditor, Washington Gas Light Company, Washington, D.C.

1985-1988 - Information Center Consultant, Washington Gas Light Company, Washington, D.C.

1988- - Programmer/Analyst, Washington Gas Light Company, Washington, D.C.

Sheila Norris

V. BIBLIOGRAPHY

[Banerjee 87]	Jay Banerjee et al. "Data Model Issues for Object-Oriented Applications." <u>ACM Transactions on Office Information Systems</u> Vol. 5 No.1 January 1987: 3-26.
[Bapa Rao 86]	D.V.Bapa Rao. "An Object-Oriented Framework for Modelling Design Data." <u>Proceedings of the 1986 International Workshop on Object-Oriented Database Systems</u> September 23-26, 1986, Pacific Grove California. IEEE Computer Society Press: 232.
[Blaha 88]	Michael R. Blaha, William J. Premerlani, and James J. Rumbaugh. "Relational Database Design Using an Object-Oriented Methodology." <u>Communications of the ACM</u> April 1988 Vol. 31 No. 4: 414-427.
[Bretl 89]	Robert Bretl et al. "The GemStone Data Management System." <u>Readings in Object-Oriented Database Systems</u>. Eds. Stanley B. Zdonik and David Maier. San Mateo, California: Morgan Kaufman Publishers, Inc., 1990. 283-308.
[Carey 88]	Michael J. Carey, David J. Dewitt, and Scott L. Vanderberg. "A Data Model and Query Language for Exodus." <u>ACM SIGMOD</u> Vol. 17 No. 3 1988: 413-123.
[Codd 70]	E.F. Codd. "A Relational Model of Data for Large Shared Data Banks." <u>Communications of the ACM</u> Vol. 13 No. 6 (June 1970): 377-387.

[Codd 79]	E.F. Codd. "Extending the Database Relational Model to Capture More Meaning." <u>ACM Transactions on Database Systems</u> Vol. 4 Number 4 (December 1979): 397-434.
[Date 86]	C.J. Date. <u>An Introduction to Database Systems</u>, 4th ed. Reading, Massachusetts: Addison-Wesley Co., 1986, Vol 1.
[Diedrich 89]	Jim Diedrich and Jack Milton. "Objects, Messages, and Rules in Database Design." <u>Object-Oriented Concepts, Databases, and Applications</u>. Eds. Won Kim and Frederick H. Lochovsky. New York, New York: ACM Press, 1989. 177-197.
[Fishman 89]	D.H. Fishman et al. "Overview of the IRIS DBMS." <u>Object-Oriented Concepts, Databases, and Applications</u>. Eds. Won Kim and Frederick H. Lochovsky. New York, New York: ACM Press, 1989. 219-250.
[Hudson 87]	Scott E. Hudson and Roger King. "Object-Oriented Support for Software Environments." <u>Proceedings of the ACM Special Interest Group on Management of Data</u> May 27-29 1987 San Francisco. Eds. Umeshwar Dayal and Irv Traigar. ACM Press: Vol. 16 No. 3 (December 1987). 491-501.
[Hudson 89]	Scott E. Hudson and Roger King. "Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System." <u>ACM Transactions on Database Systems</u> Vol. 14 No. 3 (September 1989). 291-321.

[Kent 89]	William Kent. "An Overview of the Versioning Problem." <u>ACM SIGMOD</u> 1989 Vol. 18 Number 2. 5-7.
[Khoshafian 90]	Setrag N. Khoshafian and George P. Copeland. "Object Identity." <u>Readings in Object-Oriented Database Systems</u>. Stanley B. Zdonik and David Maier, ed. San Mateo, California: Morgan Kaufman Publishers, Inc., 1990. 37-46.
[Kim 87]	Won Kim, Jay Banerjee, Hong-Tai Chou. "Composite Object Support in an Object-Oriented Database System." <u>Proceedings of 1987 OOPSLA</u>, October 4-8, 1987. ACM Press.
[Kim 89a]	Won Kim et al. "Features of the ORION Object-Oriented Database Systems." <u>Object-Oriented Concepts, Databases, and Applications</u>. Eds. Won Kim and Frederick H. Lochovsky. New York, New York: ACM Press, 1989. 251-282.
[Korth 90]	Henry F. Korth, Won Kim, and Francois Bancilhon. "On Long-Duration CAD Transactions." <u>Readings in Object-Oriented Database Systems</u>. Stanley B. Zdonik and David Maier, ed. San Mateo, California: Morgan Kaufman Publishers, Inc., 1990. 408-431.
[Nierstrasz 79]	Oscar Nierstrasz. "A Survey of Object-Oriented Concepts." <u>Object-Oriented Concepts, Databases, and Applications</u>. Eds. Won Kim and Frederick H. Lochovsky. New York, New York: ACM Press, 1989. 3-21.

[Rowe 90]	Lawrence A. Rowe and Michael R. Stonebraker. "The Postgres Data Model." Readings in Object-Oriented Database Systems. Eds. Stanley B. Zdonik and David Maier. San Mateo, California: Morgan Kaufman Publishers, Inc., 1990. 461-473.
[Schwarz 86]	P. Schwarz et al. "Extensibility in the Starburst Database System." Proceedings of the 1986 International Workshop on Object-Oriented Database Systems, Pacific Grove, California, September 23-26, 1986. Eds. Klaus Dittrich and Umeshwar Dayal. IEEE Computer Society Press. 85-92.
[Snyder 90]	Alan Snyder. "Encapsulation and Inheritance in Object-Oriented Programming Languages." Readings in Object-Oriented Database Systems. Stanley B. Zdonik and David Maier, ed. San Mateo, California: Morgan Kaufman Publishers, Inc., 1990. 84-91.
[Stonebraker 86a]	Michael Stonebaker. "Inclusion of New Types in Relational Data Base Systems." Proceedings of the International Conference on Data Engineering, Los Angeles, California, February 5-7, 1986. Washington, D.C.: Computer Society Press.
[Stonebraker 86b]	Michael Stonebraker and Lawrence A. Rowe. "The Design of Postgres." ACM Proceedings of the International Conference on Management of Data, Washington, D.C., May 28-30, 1986. Vol. 15 No. 2 (June 1986). ACM Press.

[Teorey 86] Toby J. Teorey, Dongquing Yang, and James P. Fry.
"A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model."
Computing Surveys Vol. 18 No. 2 (June 1986):
197-222.

[Tsichritzis 82] D.C. Tsichritzis and O.M. Nierstrasz.
"Fitting Round Objects Into Square Databases."
Proceedings of the European Conference on Object-Oriented Programming, Oslo, Norway, August 1988.
Eds. G. Goos and J. Hartmanis.
Berlin, Heidelberg: Springer-Verlag.
283-299.

[Ullman 82] Jeffrey D. Ullman.
Principles of Database Systems, 2nd ed.
Rockville, Maryland: Computer Science Press,
1982.

[Weiser 89] Stephen P. Weiser and Frederick H. Lochovsky.
"OZ+: An Object-Oriented Database System."
Object-Oriented Concepts, Databases, and Applications.
Eds. Won Kim and Frederick H. Lochovsky.
New York, New York: ACM Press,
1989.
309-337.

[Zaniolo 90] Carlo Zaniolo.
"The Database Language Gem."
Readings in Object-Oriented Database Systems.
Stanley B. Zdonik and David Maier, ed.
San Mateo, California: Morgan Kaufman Publishers, Inc.,
1990.
449-460.

[Zdonik 84]	Stanley B. Zdonik. "Object Management System Concepts." Proceedings of the Second ACM-SIGOA Conference on Office Information Systems June 25-27 1984 Toronto Canada. Ed. Clarence A. Ellis. ACM Press Vol.5 Numbers 1-2. 13-19.
[Zdonik 90]	Stanley B. Zdonik and David Maier, ed. Readings in Object-Oriented Database Systems. San Mateo, California: Morgan Kaufman Publishers, Inc., 1990.